Analysis of Energy and Performance of RDMA-based Data Access Patterns

Siddhartha Jana
University of Houston, Texas
sidjana@cs.uh.edu

Joseph Schuchart
Technische Universität Dresden
joseph.schuchart@tu-dresden.de

Barbara Chapman
University of Houston, Texas
chapman@cs.uh.edu

ABSTRACT
One of the factors associated with the usability of distributed programming models in exascale machines is the energy and power cost associated with data movement across large-scale systems. PGAS implementations provide users with explicit interfaces for one-sided transfers to remote processes. However, a number of factors across the software stack have the potential of significantly impacting the energy signatures of communication-intensive applications that rely on such transfers. Performance characteristics like the use of non-blocking communication, the actual count of number of initiated transfers, the size of data payload packed within each transfer, as well as the use of pinned-down user buffers, all contribute to this impact.

In this paper, we discuss a number of RDMA-based communication patterns that are frequently incorporated within applications and communication libraries and, that have the potential of significantly impacting the energy and performance characteristics. We present an empirical study of the potential energy savings achievable by studying the impact on the CPU and DRAM. Since performance is a major criteria for PGAS programming models, we use the energy-delay product as a metric to justify the feasibility of these transformations.

We hope that this work motivates the incorporation of energy-based metrics for fine tuning PGAS implementations.

Categories and Subject Descriptors
D.1.3 [Software]: PROGRAMMING TECHNIQUES—Concurrent Programming – Distributed Programming

General Terms
Performance

Keywords
Energy efficiency, Remote data transfers, PGAS Code Transformations, Data Access Patterns

1. INTRODUCTION
One of the primary challenges on the pathway to Exascale Computing is the 20MW power consumption envelope established by the U.S. Department of Energy’s Exascale Initiative Steering Committee [19]. The direct outcome of this has been a rising concern about the energy and power consumption of large-scale applications that rely on various communication libraries for data movement across distributed systems. Extreme Scale reports suggest a rise in the energy consumption during data movement off-chip as compared to on-chip. Our past work on distributed memory models like MPI[10] and PGAS models like OpenSHMEM[11] have shown that the design constraints imposed by both, the communication model, as well as, the underlying hardware, have a significant impact on the energy signatures of individual data transfers.

In this work, we extend this study to incorporate the impact of design of communication patterns characterized by multiple occurrences of such transfers. In case of PGAS models, the variation of this impact on the energy and the performance can be attributed to the flexibility of decoupling synchronization costs from the actual transfer of the data-payload. We discuss a number of one-sided based communication patterns and perform an empirical analysis of the maximum possible savings that may be obtained while choosing one access pattern over the other. These also motivate the need for static or dynamic transformations among these communication techniques. We evaluate some well known techniques like aggregating contents of source buffers of multiple remote write operations, using non-blocking data transfer semantics, using pinned-down buffers, and managing the size of data payload packed within each transfer. We present empirical results that indicate that the savings (in terms of performance and energy) obtained through such techniques varies significantly and there is plenty of opportunity for system programmers to tune for energy-efficient implementations of PGAS models.

In short, the main contributions of this paper are:

• Discussion of a number of factors characterizing individual data transfers that have the potential of impacting the energy signatures of PGAS applications.
• Discussion of multiple communication patterns that incorporate these factors while servicing data transfers.

• Empirical evidence of the feasibility of adopting such techniques for fine tuning not only the performance but also the energy efficiency of applications. This is presented in terms of the reduction in CPU energy consumption, DRAM energy consumption, communication latency, and the energy-delay product (or EDP).

We layout our study as follows. Section 2 provides a discussion on past related work. In Section 3 we discuss the various characteristics within PGAS communication kernels that have an impact on the energy and latency cost of applications. In Section 4 we define a data access pattern in terms of some basic elements and list a small subset of such patterns. This discussion is followed by some examples of transformations that have the potential of energy savings. Our experimental setup for the analysis of these transformations is described in Section 5. Following this, we back the claim of potential energy savings by present empirical results to back the claim of significance numbers are presented in Section 6.

We hope that this work motivates the incorporation of “energy-based” metrics while designing energy efficient PGAS implementations.

2. RELATED WORK

Proposals like Thrifty [20] have been initiated to pursue research efforts towards redesigning the complete computing stack. The goal of such efforts is directed towards building power-aware Exascale platforms.

Past efforts towards understanding and managing the power consumption trends of applications have been significant. One of the static based approaches for managing power consumption by processes is for the compiler to evaluate a program and determine sections within the code where the energy consumption profile changes. This knowledge in the form of power management hints can then be conveyed to the runtime to adjust the voltage/frequency scaling of applications [1]. Korthikanti and Agha [13] study the power consumption behavior of shared memory architectures while handling applications with different problem sizes. Li et al. [15] use DCT and DVFS techniques to study the opportunities of reducing power consumption of hybrid MPI-OpenMP applications.

There has been a great deal of research in managing the energy consumption of applications. Most of these efforts target energy-based optimizations for applications running in a shared memory environment. The maximum impact on the energy savings in such platforms are governed by the avoidance of penalty due to cache misses and memory-intensive operations. For example, Rahman et al. [18] propose reducing power consumption in scientific applications by decreasing the number of active threads and fine-tuning cache blocking and loop unrolling factors to achieve efficient execution. Research efforts show that power bottlenecks are common in case of “disagreements” between the application activity and the system power consumption and quite often the source of inefficiency can be tracked down to the use of power-hungry busy-waits [2, 3, 5].

Barreda et al. [4] discuss work on a Framework for aposteriori detection of power-sinks in the form of discrepancies between the application activity and the CPU C-states. Choi et al. [7] explore opportunities of using DVFS in case of memory intensive phases of applications. Their approach relies on prediction of this intensity by dynamically measuring the ratio of off-chip versus on-chip accesses.

The work closest to our focus are those by Kandemir et al. [12], Vishnu et al. [22], and Venkatesh et al. [21]. Kandemir et al. [12] discuss static based techniques like traditional data flow analysis and polyhedral algebra to detect redundant communications and unwanted synchronizations in HPF-like languages. Vishnu et al. [22] exploit voltage frequency scaling and interrupt-based methods to achieve energy savings during remote memory operations. They implement this technique in ARMCI [17]. The energy savings discussed in this work only target individual data transfer operations. Venkatesh et al. [21] discuss techniques of energy measurement of MPI-based data transfers using Intel’s RAPL scheme. Energy readings of point-to-point and collective operations are discussed. However, these efforts do not take into account the impact of multiple factors across the hardware and software stack. As we discuss in this paper, the cost of an independent data transfer construct is dependent on its semantics and the data access pattern it participates in. The following sections discuss a number of similar factors and present analysis of empirical results that are significantly impacted by them.

3. DESIGN FACTORS IMPACTING COMMUNICATION ENERGY COSTS

This section describes some application-level design factors that have the potential of impacting the energy signatures of communication-intensive kernels. While these factors are controllable at the user-level, their use directly impacts the behavior of the underlying communication library.

At a higher level, we categorize these on the basis of -(a) properties of the communication kernel, and (b) properties of individual data transfers

3.1 Properties of the Communication Kernel

The total size of the payload being transferred. In case of communication-intensive kernels, past work indicates that the total size of all the user buffers participating in RDMA operations have a direct impact on the energy consumption [21, 22, 11]. Since this metric impacts the memory footprint of the application, it is essential to incorporate this metric in empirical studies.

1 It must be noted that the significance of the impact of such a metric depends on the actual ratio of the number of local compute-based operations to those servicing remote transfers. This dependence on the “intensity” of a communication kernel is in alignment with similar empirical and energy model studies for shared-memory systems [6].
The number of explicitly initiated data transfers. While the payload size associated with data movement is important, the overhead associated with the software stack that services the transfer of the payload is equally significant. Therefore, one of the crucial factors that needs to be considered while evaluating energy and performance costs is the number of explicitly initiated data transfers. This metric takes multiple forms across the software stack:

- At the application level, this metric typically corresponds to the number of discrete user buffers used to design a communication pattern. The exact count of such buffers and their actual size is dependent on the application’s problem size and the algorithm design.

- At the data transfer layer, the impact of this metric supplements the impact of completion semantics of RDMA transfers. For example, in case of non-blocking remote write operations, this metric corresponds to the number of outstanding in-progress PUTs. In such cases, the energy and latency costs are impacted not only by the cost of servicing the actual transfers, but also that of managing and polling for the status of multiple communication handlers.

- At the bytes transfer layer, the bandwidth and the message rate are dictated by the constraints imposed by the underlying interconnect and physical layer. Due to this limit, this metric also corresponds to the actual number of chunks that the middleware divides the user buffer, before transferring its contents over the network.

For the rest of the text, we refer to this metric as “fragments” or “chunks”.

3.2 Properties of the Individual Data Transfers

The data-transfer completion semantics. Most modern interconnects support non-blocking transfers of data between the local and remote memories. The latency due to such remote transfers may therefore be overlapped by the available computation. This ensures efficient use of CPU cycles that would otherwise be busy polling for the completion status of otherwise blocking transfers. However, the use of non-blocking transfers comes with the price of: (a) having to manage multiple communication handlers, and (b) the count of the number of in-progress transfers. This management of large number of such transfers might lead to an increase in the participation of the CPU, thereby increasing the energy consumption.

The contiguity of the data-buffers in memory. While handling small to medium sized transfers, an application developer or the PGAS implementation itself may exploit the peak bandwidth of the underlying interconnect by merging multiple non-contiguous source buffers into a single contiguous chunk before sending the contents across the network.

This tactic is well-established among PGAS implementations which support strided, indexed, or vectorized transfers[17]. However, one has to be wary of the latency and the energy cost associated with such mechanisms due to (a) the impact of local memcpy/s which are CPU and DRAM intensive, and (b) the maximum achievable bandwidth of the underlying interconnect. The benefits therefore depend on the extent of hardware support and the amount of computation available for overlapping the latency associated with bulk transfers.

The registration status of the source buffers with an RDMA-capable NIC. PGAS implementations built on top of OS-bypass mechanisms require the virtual-to-physical address mapping to be pinned down. This pinned region is registered with the NIC to enable RDMA-based accesses. If the application programmer uses a source buffer that is not pinned to the memory, a PGAS implementation typically performs a local copy of the contents of the buffer to pre-registered memory locations. As shown in further sections, such local memory copies are CPU and DRAM intensive and their cost is proportional to the size of the copied contents.

4. CODE TRANSFORMATIONS THAT IMPACT ENERGY CONSUMPTION

In order to evaluate the impact of the elimination of cost factors discussed in the Section 3 we designed a number of microbenchmarks simulating different possible data access patterns using one-sided constructs. We discuss these next. This is followed by a discussion of how the different cost factors may be eliminated by transforming an access pattern from one form to another.

The results presented in this work are intended to motivate such transformations using either static or dynamic approaches. It must be noted that in real world applications, the feasibility of switching such transformations would be constrained by a number of other factors like data dependencies, algorithm design, the memory model, the communication model, etc. The discussions here and the empirical results in Section 5 are therefore aimed at aiding the reader in obtaining an optimistic estimate of the potential possible energy savings.

4.1 Data Access Patterns

In order to design multiple data access patterns within a communication kernel, we needed to identify a set of design elements, based on which any one-sided communication-intensive pattern may be modeled. These “design elements” correspond to different phases over which a remote transfer may be built upon.

4.1.1 Design Elements

RDMA Write operations (or PUTs) in PGAS implementations may be divided into the following basic phases:

2 An alternative to performing local memory copy operations is to dynamically register memory locations. However this is a very expensive operation[16,23] and is used while handling only bulk-sized buffers. The study of the impact of this metric is out of scope of this work.
4.1.2 Examples

We discuss some examples of data access patterns that were designed as a combination of the basic elements discussed above. Figure 1 illustrates these patterns, the impact of which, are discussed later in Section 4.

...PPQ : Having multiple consecutive PUTs followed by a single quiet takes into account the overhead of maintaining multiple handlers for non-blocking PUTs and polling for their completion.

...MPMPQ : Having every non-blocking PUT be preceded by a memory copy operation of the contents of the source buffer to a registered buffer takes into account the impact of using registered source buffers. After all the memory copies and the PUTs, this pattern ends with a single quiet operation, thereby guaranteeing completion of the data transfer.

...PQQ : Having every PUT be immediately followed by a quiet takes into account the impact of multiple blocking remote PUTs.

...MPQQMPQ : This benchmark represents the worst case which is a combination of all the expensive factors described above. Each PUT operation is preceded by a memcpy() and is followed by the quiet operation.

...MMPQAMM : To study the impact of the overhead due to aggregation of discrete user buffers, a number of factors need to be taken into consideration. These include: the cost of using multiple memcpy()s at the sender’s side to copy data from the user-buffer into a pinned-down source buffer, the cost of actual transfer of the buffer contents to the remote process (using a single PUT), the cost of checking for remote completion of the transfer by the sender (a single quiet), the cost of signaling the completion of the transfer to the receiver process (a single atomic operation), the cost due to polling for the completion-signal by the receiver, and the cost at the receiver’s side to copy back the contents from the destination buffer back to the final destination user buffers (using multiple memcpy()s). It must be noted that unlike the above patterns where the number of PUTs is equal to the number of user buffers, this pattern contains a single PUT following as many memory copy operations as the number of user buffers.

4.2 Transformations of Data Access Patterns

Figure 2 illustrates the set of microbenchmarks that were evaluated and the relation between them. The edges connecting the nodes of the graph depict different code transformations, the impact of which are discussed in Section 4.

\(^3\)A note on the nomenclature used: A repetition of a substring in each pattern name corresponds to a discrete user buffer. e.g. Each ‘MP’ in ...MPMPQ corresponds to operations over a different fragment at successive addresses in the heap. The actual count of this repetition i.e. the number of fragments, corresponds to the number of disjoint user buffers over which the access pattern operates.

\(^4\)A note on the design of the microbenchmarks: Obtaining steady energy readings require running the synthetic microbenchmarks for large number of iterations. To avoid a data access pattern from falling prey to caching effects from past runs, it is essential to clear the contents of the cache before the start of each iteration.
We study the impact of four different OpenSHMEM code transformations that take into account the factors discussed in Section 3. Using the nomenclature above, we describe these transformations below:

- **Impact of using pinned source buffers**: The impact of using unpinned source buffers and additional `memcpy()` operations may be studied by using globally visible or symmetric user-buffers, i.e. $MPQMPQ \rightarrow PQPQ$ and $MPMPQ \rightarrow PPQ$.

- **Impact of using non-blocking remote transfers**: The possible cost-savings associated with converting blocking remote write operations to non-blocking ones may be studied by eliminating unnecessary calls to `quiet` after each PUT, i.e., $MPQMPQ \rightarrow MPMPQ$ and $PQPQ \rightarrow PPQ$.

- **Cost of aggregating user-buffers**: The impact of using multiple memory copy operations to aggregate data into pinned memory instead of explicitly issuing multiple PUTs may be evaluated through the transformation $PPQ \rightarrow MPMPQAMM$.

- **Cost of reducing the data-payload**: While all the above transformations are dependent on the characteristics of the data access pattern within a communication kernel, this transformation deals with the size of the data-payload as dictated by the input problem size. The impact of the data-payload size can be analyzed by studying the same test-case $MPQMPQ$ with different transfer-payload sizes.

It must be noted that many other transformations and their combinations such as $MPQMPQ \rightarrow MPMPQAMM$ are also possible. However, since our scope lies on studying each of transformations independently, we do not discuss such cases here. Their impact may be compounded over more than one transformations listed above.

### 5. EXPERIMENTAL SETUP

#### 5.1 Test-bed Characteristics

The details of the experimental setup are listed in Table 1. All experiments were conducted using two OpenSHMEM processes (PEs) where each process was launched and bound to one of the cores of an Intel Sandy Bridge processor on a separate node. These two nodes are prototypes for an upcoming installation at the University of Technology Dresden, which are instrumented for fine-grained accurate power measurement. Each node has instrumented voltage regulators (VRs) that are sampled with a sampling frequency of 1 KHz for both sockets and the four voltage lanes of the DIMMs on board. With the help of an FPGA, a digital filter is applied to smooth the samples. Furthermore, a linear correction is applied to the measurement data coming from the VRs in order to ensure an error margin not exceeding 3%. Our study was aimed at performing a fine-grained analysis of the impact on two main components that dictate the energy and power consumption of a system - the CPU and the DRAM. We limit our scope to the study of these two components. For studies on large-scale systems, the contribution of the interconnect and the network topologies becomes crucial, as highlighted by recent study[14].

#### 5.2 Software Stack Characteristics

The empirical results presented in this paper were obtained using synthetic OpenSHMEM microbenchmarks. They design of these benchmarks were based on the line diagrams depicted in Figure 1. The impact of the PGAS-based operations listed in [4,11] were studied using OpenSHMEM's constructs like `shmem_putmem()` and `shmempd_wait()` interfaces. The OpenSHMEM implementation used was Mellanox Scalable SHMEM (ver-2.2) over OpenFabrics Byte Transport Layer.

### 6. EMPIRICAL RESULTS

While one of the primary purposes of this paper is to discuss the potential savings achievable by transformation of a certain data access pattern to another, it is essential to understand the behavior of independent patterns themselves. This can be achieved by analyzing the energy costs, latency,
message rate, and bandwidth of each of these patterns. Figure 3 illustrates these transfer characteristics of the communication patterns while servicing a 0.5MB remote write operation using different number of PUT operations (represented as “#Fragments” on the x-axis). We observe that access-patterns with different blocking semantics have significant change in energy and performance traits beyond 256 PUTs (<2KB/PUT). Communication using blocking semantics (MPQMPQ and PPQ) have the highest energy and latency cost. This is accompanied with a lower bandwidth and message rate. This trend can be attributed to the penalty associated with polling-based operations and additional memory management necessary to ensure remote completion of the transfers. This impact on the energy and performance is reduced due to the use of non-blocking semantics (MPMPQ and PPQ). With the number of PUTs greater than 256 (<2KB/PUT), aggregation of data buffers (MMPQAMM) lead to minimal energy consumption and latency, and a sharp rise in the bandwidth and the number of buffers transferred per second.

Moreover, for all data access patterns except MMPQAMM, the message-rate becomes limited beyond 64 PUTs (<8KB/PUT) and is accompanied with a steady drop in the bandwidth of the transfer. This corresponds to the software overhead due to multiple explicitly initiated PUTs.

While these raw values provide an overview of the behavior of the data patterns, they do not present a fine-grained insight into the impact on the CPU and the DRAM. Moreover, they do not present a clear indication of the potential savings due to the factors discussed in Section 3. To address this, we present a detailed study using the transformations discussed in Section 4. These are shown in Figures 4, 5, and 6. These figures illustrate the impact of the transformations on various cost metrics.

In our case, the cost “metrics” studied are:

- Energy consumption by the CPU
- Energy consumption by the DRAM
- Latency of the transfer
- EDP or Energy Delay Product

The “impact” of each cost metric is calculated in terms of the percent reduction in one of the above metrics. If a transformation $T$ is applied on a data access pattern $C_{initial}$ such that: $T(C_{initial}) \rightarrow C_{final}$, then the impact of $T$ in terms of percent reduction in a cost-metric $M$ may be calculated as:

$$I = \frac{M(C_{initial}) - M(C_{final})}{M(C_{initial})} \times 100$$

While CMOS circuits have the ability to trade performance for energy savings, it becomes challenging to optimize both simultaneously. The EDP, first proposed by Horowitz [8, 9], takes into account both the energy and the time costs in an implementation-neutral manner. For cases, where energy and performance have equal importance, this metric can be calculated as a product of the energy consumed and the time taken. For more complicated cases, where performance is given a higher priority, the weight of the “delay” factor is increased by squaring or cubing it [14].

---

*In case of MMPQAMM, all the buffers are serviced by a single PUT operation, irrespective of the number of user buffers. Thus the metric - message rate corresponds to \#User-buffers/sec.*
For all of these experiments, the graphs depict the values of various metrics as measured at the compute node servicing the active sender processes responsible for initiating the remote write operations. We restrict our discussion to study the behavior of this process and not the passive receiver process.

It must be noted that the energy consumption of a passive process that’s polling at a barrier, waiting for the completion of a transfer, cannot be ignored while performing large scale studies of distributed applications. In fact, our past study[11] indicates that the energy consumption increases proportionally with the time and its scale is very high. However, since the polling activity corresponds to a constant power consumption, it can be safely ignored in the following discussions that focus on the impact due to the remote data access patterns.

6.1 Impact of Using Pinned Buffers
From Figure 4, we observe that the impact on (or, the percent of reduction in) the CPU energy consumption and the latency is as high as 20% in case of bulk transfers. This is not surprising, as this type of transformation results in elimination of unwanted memory copy operations of bulk buffers, which directly benefit the energy cost and the latency. This elimination of bulk memory copy operations leads to DRAM energy savings as high as 40%. The impact of this transformation however drops to less than 5% in terms of CPU energy and almost zero in case of latency. This downward trend is observable especially when the number of PUTs increases beyond 512 (i.e. buffer size < 1KB per PUT). This condition corresponds to the case where the CPU energy cost of small memory copy is negligible and the message rate reaches its threshold (see Figure 3).

Influence of other cost factors:. We see that there exists a variation in the impact based on the blocking semantics of the PUT operations. The plots in the figure depict these two possible outcomes as “Blocking PUTs: MPQMQ → PQPQ” and “Non-Blocking PUTs: MPMPQ → PPQ”. The primary observation with regards to the DRAM energy consumption is that there is an overall lesser impact of this transformation on blocking PUTs when compared with non-blocking PUTs. An interesting observation is the oscillating trend in DRAM energy savings in case of using non-blocking PUTs. This was surprising because the total size of the data payload being handled across all the data points remains constant (0.5MB) and the primary source of DRAM energy savings in this transformation is the elimination of local memory copy operations. We are currently investigating the reason for this trend.

6.2 Impact of Using Non-Blocking Remote Transfers:
From Figure 5, we observe that the impact of replacing blocking transfers by non-blocking versions is significant in terms of reduction in CPU energy, latency, and the energy-delay product. As shown, the positive impact on the energy and the latency rises with an increase in the number
Figure 5: Impact of transforming multiple blocking operations to non-blocking

Figure 6: Impact of aggregation of multiple data buffer
of discrete PUTs and hits a limit (80%) when this count rises beyond 256. This can be attributed to the fact that the benefits of launching multiple non-blocking transfers is overshadowed by the cost of ensuring completion of these transfers (during the quiet operation). The benefits on the energy-delay product is significant. The 80% reduction in CPU energy and latency corresponds to an improvement in energy delay product by almost 95%.

**Influence of other cost factors.** We see that there is very little difference between the cases corresponding to using pre-registered and non-registered data buffers. The plots in the figure depict these two possible outcomes as “Unpinned Source: MMP QAMM” and “Pinned Source: PQP Q → PPQ”.

### 6.3 Impact of Aggregation of Buffers:

Unlike other data access patterns discussed in the text, an access pattern like MMPQAMM corresponds to an active participation by both the sender and the receiver. Moreover the RDMA-based data transfer is limited to a single PUT operation. The cost associated with handling multiple user buffers is dependent solely on the cost of local memory copy operations.

Figure 3 depicts the impact of converting a pattern like PPQ to MMPQAMM. The observations are described below.

**For bulk transfers (#Fragments≤1024):** For this case, a pattern like PPQ is characterized by few bulk PUTs directly from the source buffer to that target. The transformed pattern MMPQAMM is characterized by bulk local memory copy operations first by the sender, and later on by the receiver. The latter pattern significantly raises the CPU and DRAM energy consumption. Moreover, this phase corresponds to the peak bandwidth achievable using PPQ. Thus, we see a negative impact on the energy metrics (about -25% on CPU and -125% on DRAM) and the latency (-25%). This negative impact amortizes any potential energy savings achievable through the use of a single bulk blocking PUT.

**For small transfers (#Fragments>1024):** We observe that the overall CPU and DRAM energy savings achieved using this transformation increases with the count of discrete source data buffers (fragmentation). Besides the obvious elimination of the software overhead of multiple PUTs and handling multiple in-progress transfers in PPQ, the high energy savings may be attributed to this pattern’s limiting message-rate and dropping bandwidth, as shown in Figure 3.

To summarize, we learn that with no sufficient overlap for transfers of bulk-sized buffers, initiating a non-blocking RDMA operation does not yield much benefit. Additionally, we observed that despite data access patterns like aggregation have positive impact on energy and performance based metrics for large number of small-sized user buffers, its adoption for bulk-sized messages becomes inefficient. Similarly, enforcing the use of pinned-down memory for small-sized user buffers is not beneficial.

### 7. CONCLUSION

In this paper, we established the notion that the design of data access patterns play a critical role in impacting the energy profiles of communication-intensive PGAS applications. We investigated a number of factors that affect the energy cost of a process initiating a remote data transfer. These include – the continuity of the data buffers in the memory, the total size of the payload being transferred, the registration status of the source buffers, the completion semantics of the data transfer operations, etc. For a fixed size of data-payload that is transferred to a remote process, the extent of impact of these factors depends on the number of explicitly initiated data transfers.

We investigated the impact of different pattern transformation techniques on the energy and performance characteristics of communication intensive kernels, using: registered memory buffers (upto 40% EDP savings), non-blocking operations (upto 97% EDP savings), and aggregation of source buffers (-70% to +98% EDP savings). Some of the lessons learned include -(a) Energy savings achieved by using pinned-down source buffers reduces with a rise in the number of explicitly initiated PUT operations, (b) Energy savings due to the use of non-blocking semantics is higher for smaller sized transfers; such savings hit a limit due to additional overhead of management of multiple outstanding remote transfers, (c) Aggregating bulk-sized buffers into contiguous memory locations has a negative impact on the energy savings, the latency and the energy-delay product. Using multiple smaller transfers tend to benefit significantly in terms of such savings.

### 8. ACKNOWLEDGMENTS

This work has been supported through resources from the High Definition Energy Efficiency Monitoring (HDEEM) research project at the Center for Information Services and High Performance Computing (ZIH) at the Technische Universität Dresden. Special thanks are due to Daniel Hackenberg, Daniel Mollka, Robert Schöne, and Thomas Ilsche for their help and guidance while using these resources.

### 9. REFERENCES


